In the Claims

1. (Currently amended) A method of generating executable code for a configurable microprocessor architecture which has an instruction set and a plurality of physical data path connections between execution units, comprising ~~whereby~~:

(a) situating ~~there are~~ individual registers in the data paths between execution units;

(b) updating said ~~the update of these~~ registers by explicit management ~~is explicitly managed~~ from the instruction set of the processor; ~~and~~

(c) ~~the~~ reading and writing of ~~those~~ said registers, whereby said reading and writing is represented in a data flow graph representation of [[the]] a program being targeted to the processor;

(d) representing dependencies between reads and writes to registers as edges in the graph;

(e) detecting, by the presence of cycles within the graph, graphs that invalidate the register access dependency rules;

(f) performing cyclic detection after each individual register allocation performed on the graph to detect an allocation which is illegal; and

(g) using the cycle detection mechanism to force alternative allocation decisions to be made to avoid cyclic graphs and direct the allocation to an acyclic and thus legal allocation.

2. (original) The method according to claim 1 whereby the number of registers associated with particular execution units is configurable.

3. (original) The method according to claim 1 whereby the target architecture is specified in an input file.

4. (original) The method according to claim 1 whereby certain units may only be connected to a subset of other execution units in the architecture.

5. (original) The method according to claim 4 whereby there is a central register file but only a subset of the execution units in the system can directly communicate with it.

6. (original) The method according to claim 3 whereby the input program is initially represented as a sequence of operations that can be performed by execution units present in the target architecture.

7. (original) The method according to claim 6 whereby the inputs to and results from these basic instructions may be communicated via a central register file.

8. (original) The method according to claim 7 whereby the code sequence may be optimised to reduce the communication required with the central register file.

9. (original) The method according to claim 8 whereby multiple reads of a given register file value may be transformed into a single read with suitable communication of the same data to other consumers of the data value.

10.(original) The method according to claim 8 whereby pairs of central register file writes and reads may be transformed to use direct communication between the generating and consuming operations.

11.(original) The method according to claim 1 whereby the data flow graph is transformed so that data edges correspond to physical connections in the architecture.

12.(original) The method according to claim 11 whereby additional nodes may be inserted into the graph to represent the copying of data values where there is no physical connection corresponding to the graph data flow.

13.(Cancelled)

14.(Cancelled)

15.(original) The method according to claim 1 whereby an idealised form of the graph is generated that assumes the availability of unrestricted connectivity in the architecture.

16.(original) The method according to claim 15 whereby the idealised form of the graph is used to influence the binding of operations to physical execution units in the architecture.

17. (original) The method according to claim 1 whereby special edges within the graph represent communication of data via a central register file.

18. (original) The method according to claim 17 whereby operations from different basic blocks may be represented in a single graph.

19. (original) The method according to claim 1 whereby individual operations in the graph are bound to particular execution unit instances.

20. (original) The method according to claim 19 whereby the unit binding uses an estimate of the delay caused by transporting operands to and results from the operation as a factor in the allocation.

21. (original) The method according to claim 20 whereby the transport cost is dependent on the structure of connectivity between the operations in the graph.

22. (original) The method according to claim 1 whereby the graph may be updated as new physical paths are added to the architecture in order to reduce the graph height to allow shorter code schedules.

23. (original) The method according to claim 1 whereby individual execution units are controlled by particular bits within the overall execution word.

24. (original) The method according to claim 23 whereby certain bits in the execution word may be used to control more than a single execution unit.

25. (original) The method according to claim 24 whereby the allocation of bits in the execution word to particular execution units is optimised as the architecture is generated.

26. (original) The method according to claim 25 whereby the usage of individual execution units is used to influence the allocation of the execution word.

27.(Previously presented) A microprocessor configured to execute code that has been generated using the method of claim 1.